Local Simplification

Christopher Lynch

INRIA Lorraine et CRIN
Campus scientifique
BP 101
54602 Villers-lès-Nancy cedex
France

Phone Number: (33) 83 59 30 12
Fax Number: (33) 83 27 83 19
Email Address: lynch@loria.fr

Proposed Running Head: **Local Simplification**

Proofs should be sent to Christopher Lynch (INRIA—address on previous page).

List of Symbols

Other than ordinary text in variously sized fonts, and in roman, italic, boldface, greek letters, ordinary mathematical symbols ($\cup$, $\in$, $\notin$, $\subseteq$, $\setminus$ (set difference), $\emptyset$, ∞, $\neg$, $\vee$, etc.) and the "at-sign" @ used in email addresses, we use the following mathematical symbols:

- Relational symbols: $\approx$ (equality in the formal language), $=$ (equality in the meta language), $\not\approx$ (negation of equality) $<$, $\leq$, $>$, $\geq$, $\not<$, $\not>$;
- Grouping symbols: $[\cdot]$, $[\cdot]_p$, $[\![\cdot]\!]$, $(\cdot)$, $\{\cdot\}$,
- Miscellaneous $\Rightarrow$, $\models$ (logical implication), $\square$ (empty clause), and sometimes we put a bar over symbols, as in $\bar{L}$.

We also use standard form for inferences, e.g.,

$$\frac{C_1 \qquad C_2}{C_3}$$

Abstract. We present a modification to the paramodulation inference system, where semantic equality and non-equality literals are stored as *local simplifiers* with each clause. The local simplifiers are created when new clauses are generated and inherited by the descendants of that clause. Then the local simplifiers can be used to perform demodulation and unit simplification, if certain conditions are satisfied. This reduces the search space of the theorem proving procedure and the length of the proofs obtained. In fact, we show that for ground SLD resolution with any selection rule, any set of clauses has a polynomial length proof. Without this technique, proofs may be exponential. We show that this process is sound, complete, and compatible with deletion rules (e.g., demodulation, subsumption, unit simplification, and tautology deletion), which do not have to be modified to preserve completeness. We also show the relationship between this technique and model elimination.

1 Introduction

The paramodulation inference system is an extension of the resolution inference system to deal with theorem proving in first order logic with equality (Robinson & Wos (1969)). Unfortunately, the paramodulation inference system can be very prolific, creating many clauses when searching for a proof. Recently, some restrictions on paramodulation have been developed to limit the search space (Bachmair & Ganzinger (1994), Bachmair et al (1995), Hsiang & Rusinowitch (1991), Nieuwenhuis & Rubio (1992), Nieuwenhuis & Rubio (1995), Peterson (1983), Pais & Peterson (1991), Zhang (1988)).

A paramodulation theorem prover can be viewed as a Knuth-Bendix completion procedure, extended to handle disjunctions of equations. Knuth-Bendix completion is fairly efficient, because each equation generated can be simplified by the other equations, thereby saving many inferences and keeping equations in a reduced form. In paramodulation inference procedures, we may simplify clauses with positive unit clauses[1], exactly as we do in completion (Wos et al (1967)). Unfortunately, the effect of this is limited, because not many unit equations exist. In this paper, we show how to increase the number of demodulations. The idea is that if a paramodulation inference has been performed using equation $s = t$, yielding equation C, then descendants of C may be simplified by $s = t$. Sometimes this is not sound and other times it is not complete. In this paper, we show exactly which simplifications can be performed, preserving soundness and completeness. We call this *local simplification*, because the simplifiers are local to clauses, and may not be used globally, as can unit equations.

In Kirchner, Kirchner & Rusinowitch (1990), it was shown how constraints could be used in theorem proving to limit the instances of a clause. Since then many papers (Bachmair et al (1995), Lynch & Snyder (1995), Nieuwenhuis & Rubio (1992), Nieuwenhuis & Rubio (1995), Nieuwenhuis & Rubio (1994), Vigneron (1994)) have appeared which use constraints to restrict the search space. These papers show that other restrictions on the search space can be represented as constraints. Constraints are built up as inferences are performed and used to remember earlier inferences. Local simplifiers are similar to constraints in that they are inherited in the same way. In contrast, constraints are a way of restricting the number of instances of a clause, but local simplifiers are not. Therefore, local simplifiers have an advantage over constraints in terms of redundancy rules, as we will show.

Local Simplification works as follows When an inference is performed, an equation is saved as an ancestor literal (as in Model Elimination (Loveland (1968), Loveland (1978)), along with some conditions. This is called a *local simplifier*. When the conditions are met, that equation can be used to simplify the clause. This may limit the search space, but in contrast with other restrictions it may also shorten the refutation. An advantage our local simplifiers have over other constrained theorem provers is that they are not weakened when deletion rules, such as demodulation, unit simplification and subsumption, are performed.

[1] This is called simplification or demodulation.

To show the completeness, we use techniques from Bachmair & Ganzinger (1994) to remove redundant clauses. This paper is evidence to the power of these redundancy techniques, because we can show our completeness results by appealing to the proof in that paper.

As a motivation of the need for local simplification, consider the following resolution refutation proof.

$$\cfrac{\neg R \qquad \cfrac{R \vee \neg P \qquad \cfrac{P \vee \neg Q \qquad \cfrac{\neg R \qquad \cfrac{R \vee \neg P \qquad P \vee Q}{R \vee Q}}{Q}}{P}}{R}}{\square}$$

In this proof, the variable P has been removed from the clause $P \vee Q$ by the first resolution inference. However, it later reappears, and the same set of inferences is repeated. Local Simplification allows us to avoid that repeated set of inferences by remembering which variables are removed by resolution and then immediately removing them if they reappear. Local Simplification would result in the following proof.

$$\cfrac{P \vee \neg Q \qquad \cfrac{\neg R \qquad \cfrac{R \vee \neg P \qquad P \vee Q}{R \vee Q \llbracket \neg P \rrbracket}}{Q \llbracket \neg P, \neg R \rrbracket}}{\cfrac{P \llbracket \neg P, \neg R, \neg Q \rrbracket}{\square}}$$

In this proof, the literals are recorded as they are resolved away, and then immediately removed when they reappear. The literals $\neg P$, $\neg Q$ and $\neg R$ are saved as local simplifiers. In the last step, the variable P reappears and is immediately removed by a local simplification step. (i.e., the clause P is replaced by the empty clause). This example shows how Local Simplification limits the size of proofs. It also reduces the search space, because the extra clauses do not need to be created. Local simplifiers contain ancestor literals as in Model Elimination (Loveland (1968), Loveland (1978)), and the local simplification step is similar to a reduction step in Model Elimination. The difference is that reduction is needed to preserve completeness and local simplification is only used to reduce the search space and the length of proofs. [2] We can also use local simplification to limit the search space of equational theorem provers and the size of paramodulation proofs as illustrated by the following example . Consider the following Paramodulation proof.

[2] We discuss more about the relationship with model elimination in the conclusion

$$\dfrac{\neg Q(b) \qquad \dfrac{\neg R \qquad \dfrac{R \vee a \approx b \qquad \dfrac{\neg P(b) \vee Q(a) \qquad \dfrac{\neg R \qquad \dfrac{R \vee a \approx b \qquad P(a)}{R \vee P(b)}}{P(b)}}{Q(a)}}{R \vee Q(b)}}{Q(b)}}{\Box}$$

In the first paramodulation inference, a is rewritten to b. In one of the descendants, a reappears. It is then necessary to perform the same sequence of inferences as before. In local simplification, we store $a \approx b$ as a local simplifier and immediately rewrite a to b whenever it appears. We call it *Local Simplification* because we can only perform this simplification locally in this clause, and not globally in other clauses. We now have the following proof.

$$\dfrac{\neg Q(b) \qquad \dfrac{\dfrac{Q(a) \vee \neg P(b) \qquad \dfrac{\neg R \qquad \dfrac{R \vee a \approx b \qquad P(a)}{R \vee P(b) \llbracket a \approx b \rrbracket}}{P(b) \llbracket a \approx b \rrbracket}}{Q(a) \llbracket a \approx b \rrbracket}}{Q(b) \llbracket a \approx b \rrbracket}}{\Box}$$

We only show the local simplifier that was used, but there are other local simplifiers that could be saved. In this example, we recorded the fact that $a \approx b$, then when a reappeared, it was rewritten to b. In other words, the clause containing a is removed as the clause containing b is added. This cannot be done in Model Elimination.

Unfortunately, Local Simplification cannot always be applied. Consider the following proof.

$$\dfrac{\neg P(b) \qquad \dfrac{\neg R(b) \qquad \dfrac{\dfrac{R(a) \vee a \approx b \qquad P(a)}{R(a) \vee P(b) \llbracket a \approx b \rrbracket}}{R(b) \vee P(b) \llbracket a \approx b \rrbracket}}{P(b) \llbracket a \approx b \rrbracket}}{\Box}$$

We record the fact that $a \approx b$ then use it to rewrite a to b in a later step. Unfortunately, our initial set of clauses: $\{P(a),\ R(a) \vee a \approx b, \neg R(b), \neg P(b)\}$ is satisfiable. Intuitively, the problem is that we used $a \approx b$ to rewrite $R(a)$ but they both came from the same clause. Any descendant of $R(a)$ would cause a similar problem. To avoid that, we save literals with each ancestor literal, which record where each clause came from. This is the only thing which is necessary to preserve soundness. It is slightly more sophisticated than the Model Elimination technique, which keeps track of the position of an ancestor literal in a clause.

Another problem if we are not careful, is that Local Simplification may not be complete. We will discuss this in a later section.

This paper is organized as follows. In section 2 we give the necessary definitions. In sections 3 and 4, we give the modifications necessary to the inference and deletion rules to handle local simplifiers. In section 5, we give the local simplification rules used to simplify clauses. Section 6 proves completeness. Section 7 is a discussion of how the local simplification rules react in combination with some well known restrictions. In SLD resolution with ground non–equational clauses, we show that every set of clauses has a linear refutation for every selection rule, whereas other goal directed inference rules have only exponential refutations of some sets of clauses with certain selection rules. This is for the non-equality case. However, we believe our results are especially powerful with equality. Section 8 shows some ways that local simplifiers can be simplified. Finally, section 9 shows the relationship with related work, including model elimination.

2 Preliminaries

We use the standard definitions of theorem proving (see Loveland (1968), Loveland (1978)).

Definition 1. A *term* is built from function symbols, constants and variables. An *atom* is a predicate symbol applied to some terms. The equality symbol is a distinguished binary predicate symbol ($\approx$), written in infix notation. For example $s \approx t$ is an equality. Negative equalities $\neg(s \approx t)$ are written as $s \not\approx t$. A *literal* is an atom or a negated atom. If L is a literal, then $\bar{L}$ is the negation of L, i.e., if $L = A$ then $\bar{L} = \neg A$ and if $L = \neg A$ then $\bar{L} = A$. An *undecorated clause* is a disjunction of literals. Sometimes clauses will be viewed as multisets. We represent clauses with the letters C and D, literals with L and M, and atoms with A and B.

We next give the definition of decorated clause as used in this paper. A decorated clause is an (undecorated) clause with some literals attached to it, which can be used to simplify the clause. These literals, which may be equations, are created when the clause is created and inherited by descendants of the clause. These literals are saved along with certain conditions that indicate when they may be used to simplify the clause, remove a literal from the clause, or remove the entire clause.

Definition 2. A *(decorated) clause* is of the form $C [\![(L_1, C_1, S_1), \cdots, (L_m, C_m, S_m)]\!]$ where C is an undecorated clause, and each (L_i, C_i, S_i) is a *local simplifier* on C. Each L_i is an *ancestor literal*, which is used for simplification of the clause. Each C_i is a disjunction of literals, and each S_i is a set of undecorated clauses. S_i and C_i determine when L_i may be used for simplification. We use variables like φ and Ψ to represent sets of local simplifiers. Since they are sets, the order is unimportant.

When we refer to a clause, we may be referring to a decorated or an undecorated clause, depending on the context.

Substitutions are defined as usual.

Definition 3. A *substitution* is a mapping from variables to terms, which is the identity on all but finitely many variables. We identify a substitution with its homomorphic extension. If σ is a substitution, then $t\sigma$ represents the result of applying σ to t. In general, $(C[\![\varphi]\!])\sigma = C\sigma[\![\varphi\sigma]\!]$. If σ and η are substitutions, then $\sigma\eta$ is defined so that $x\sigma\eta = (x\sigma)\eta$ for all variables x. We say $\sigma \leq \theta$ if there is an η such that $\sigma\eta = \theta$. We call σ a *unifier* of s and t if $s\sigma = t\sigma$. We call σ a *most general unifier* of s and t (written $mgu(s,t)$) if σ is a unifier of s and t and for all unifiers θ of s and t, $\sigma \leq \theta$.

A *multiset* is an unordered collection with possible duplicate elements. We denote the number of occurrences of an object x in a multiset S by $S(x)$. A clause is viewed as a multiset of literals. A *reduction ordering* $<$ is a well–founded ordering which is stable under context (i.e., if $s < t$ then $u[s] < u[t]$) and stable under substitution (i.e., if $s < t$ then $s\sigma < t\sigma$). Let $<$ be a reduction ordering on the literals, total on ground terms. This ordering is extended to an ordering on the clauses by identifying $<$ with its multiset extension. In other words, if S and T are finite multisets, then $S < T$ if and only if there exists an $L \in T$ such that $T(L) > S(L)$ and for all $M > L$, $S(M) = T(M)$. We say that A is maximal in a multiset S if there is no B in S such that $B > A$.

One key difference between the local simplifiers in this paper and the constraints in other papers on constrained deduction is that we do not use the local simplifiers to determine ground instances.

Definition 4. A *ground clause (resp. term)* is a clause (resp. term) without variables. A substitution σ *grounds* a clause (resp. term) C if $C\sigma$ is ground. In this paper, when we call σ a *ground substitution*, we mean that σ grounds certain terms which are obvious from the context. The set of *ground instances* of $C[\![\varphi]\!]$, $\mathrm{Gr}(C[\![\varphi]\!]) = \{C\sigma | \sigma \text{ grounds } C\}$. And $\mathrm{Gr}(S) = \{\ \mathrm{Gr}(C)\ | C \in S\}$.

The implication of this definition of ground instances is that it will not be necessary to weaken local simplifiers when performing simplification and subsumption.

We will use the notation $\models$ to mean logical implication.

Definition 5. If S is a set of ground clauses and C is a ground clause or a literal, then $S \models C$ if and only if C is true in every model of S. If T is a set of ground clauses or literals, then $S \models T$ if and only if $S \models C$ for all $C \in T$. In general, if S is a set of clauses and C is a clause or literal, then $S \models C$ if and only if $Gr(S) \models Gr(C)$. A clause C is *satisfiable* if and only if there is a model that makes C true. The empty clause $\Box$ is unsatisfiable. A set of clauses S is satisfiable if and only if some model makes all of the clauses in S true.

A clause is *redundant* if it is implied by smaller clauses. The definition of redundancy is adapted from Bachmair & Ganzinger (1994). Redundant clauses may be removed, because they are not needed for finding a proof.

Definition 6. Let T be a set of clauses. A clause C is *redundant at T in S* if for all ground substitutions σ there are clauses $D_1, \cdots, D_n \in Gr(S)$, where $n \geq 0$ such that

1. $D_1, \cdots, D_n \models C\sigma$, and
2. $\{D_j\} < T\sigma$ for all j, $1 \leq j \leq n$.

C is *redundant up to T in S* if the $<$ in condition 2 is replaced by $\leq$. For a clause D, C is *redundant at D in S* if C is redundant at $\{D\}$ in S. A clause C is *redundant in S* if C is redundant at C in S.

The local simplifiers are added when a clause is generated and inherited from the clause's ancestors. In the following definition, we show the meaning of the local simplifiers.

Definition 7. The clause $C\,[\![\,(L_1, C_1, S_1), \cdots, (L_m, C_m, S_m)\,]\!]$ is said to be *correct in S* if, for each i,

1. $C_i \subseteq C$,
2. $C_i \vee L_i$ is redundant at S_i in S, and
3. $L_i \notin C_i$.

Conditions 1 and 3 are needed for soundness. Condition 2 will be needed for completeness. It will assure us that a clause may be removed because the inference creating it is redundant. We only need to know the maximal elements of S_i so we may remove everything else from the set.

For example, the clause $P(a) \vee Q(b) \vee R(c)\,[\![\,(a \approx b, Q(b) \vee R(c), \{D, E\})\,]\!]$ is correct in S if $D = Q(b) \vee P(c)$ and $E = R(c) \vee \neg P(c) \vee a \approx b$ are clauses in S.

If a clause is ever created where the first two conditions of the definition hold, but the third condition does not (i.e., for some i's, $L_i \in C_i$), then we may replace each local simplifier of the form $(L_i, L_i \vee C', S_i)$ with (L_i, C', S_i). The first two conditions will still hold, so the clause is now correct. Therefore, when checking for correctness of a clause, we only check the first two conditions. We assume that the above transformation is performed whenever a clause is created.

3 The Inference Rules

In this section, we present the inference rules. They are the same as the inference rules in Bachmair & Ganzinger (1994), except we show how the local simplifiers are created by inference rules, and later inherited by descendants of the clause.

Each inference rule will be of the form

$$\frac{C_1 \cdots C_n}{C\sigma}$$

where $n \geq 1$. The substitution σ is the most general unifier in the inference.

This definition means that, if $C_1 \cdots C_n$ are existing clauses, then the clause $C\sigma$ must be added to the set of clauses.

Definition 8. An inference rule is *correct* if, for all S such that $C_1, \cdots, C_n \in S$ and $C_1, \cdots, C_n$ are correct in S,

1. $S \models C\sigma$, and
2. $C\sigma$ is correct in S.

To restrict the literals of a clause that may be involved in an inference, we define a selection rule to select certain literals from the clause. (see Bachmair & Ganzinger (1994), Bachmair et al (1995)). Only the selected literals may be involved in an inference.

Definition 9. A *selection rule Sel* is a function from a clause C to a subset of the literals in C. If $L \in Sel(C)$ we say that L is selected in C. A selection function is *valid* if for each C, either a negative literal is selected in C or all maximal literals of C are selected in C.

Thus, for a valid selection rule, any clause containing a negative literal only needs to select one literal, but it may be necessary to select more than one literal in a positive clause. When the inference rules are given, we assume a selection rule has been defined. A set of inference rules can be thought of as being instantiated by a selection rule. It is known that a Paramodulation inference system is complete if it is instantiated by a valid selection rule. No invalid instantiations are known to be complete.

The inference rules are now stated and proved to be correct.

Resolution is an inference rule for first order logic. It is also necessary in combination with paramodulation for first order logic with equality. Although, if all literals are encoded as equations, then resolution can be encoded by paramodulation.

Resolution

$$\frac{C \vee A\,[\![\,\varphi\,]\!] \qquad \neg A' \vee D\,[\![\,\Psi\,]\!]}{(C \vee D\,[\![\,(A, C, \{C \vee A\}), (\neg A', D, \{\neg A' \vee D\}), \varphi', \Psi'\,]\!])\sigma}$$

where $\sigma = mgu(A, A')$ [3], $A\sigma$ is selected in $(C \vee A\,[\![\,\varphi\,]\!])\sigma$ and $\neg A'\sigma$ is selected in $(\neg A' \vee D\,[\![\,\Psi\,]\!])\sigma$.

Also, φ' is the same as φ with the following exception. For any triple in φ of the form $(L_i, C' \vee A, S_i)$ (i.e., any triple where C_i contains the literal that has been resolved upon), that triple must be replaced in φ' by the triple $(L_i, C' \vee D, S_i \cup \{D \vee \neg A'\})$. This preserves the fact that $C_i \subseteq C \vee D$, because of the assumption that $L_i \notin C_i$.

Similarly, Ψ' is the same as Ψ, except that any triple in Ψ of the form $(M_i, D' \vee \neg A', T_i)$ is replaced in Ψ' by the triple $(M_i, D' \vee C, T_i \cup \{C \vee A\})$.

[3] Variables are renamed so the clauses in an inference have different variables.

Proposition 10. *Resolution is correct.*

Proof. The premises imply the conclusion, but we must also show that the conclusion is correct. First we note that the second parameter of each new local simplifier in the conclusion is a subset of the clause, because $C\sigma \subseteq (C \vee D)\sigma$ and $D\sigma \subseteq (C \vee D)\sigma$ and each local simplifier C_i or D_i has been modified so that $C_i\sigma \subseteq (C \vee D)\sigma$ and $D_i\sigma \subseteq (C \vee D)\sigma$.

Now we must check the second condition of the definition of a correct clause. Trivially the new local simplifiers satisfy the condition, because $(C \vee A)\sigma \models A\sigma \vee C\sigma$ and $(\neg A' \vee D)\sigma \models \neg A'\sigma \vee D\sigma$. In addition, we must examine each local simplifier that was modified by the inference. For each ground substitution η, there is some $S' \leq S_i\sigma\eta$ such that $S' \models (C' \vee A \vee L_i)\sigma\eta$, therefore $S' \cup \{(\neg A' \vee D)\sigma\eta\} \models C'\sigma\eta \vee D\sigma\eta \vee L_i\sigma\eta$. Similarly, there is some $T' \leq T_i\sigma\eta$ such that $T' \models (D' \vee \neg A' \vee M_i)\sigma\eta$, therefore $T' \cup \{(C \vee A)\sigma\eta\} \models D'\sigma\eta \vee C\sigma\eta \vee M_i\sigma\eta$. □

Factoring is needed in combination with resolution for completeness. However, it is only necessary to factor positive literals.

Positive Factoring

$$\frac{C \vee A \vee A' \llbracket \varphi \rrbracket}{(C \vee A \llbracket \varphi \rrbracket)\sigma}$$

where $\sigma = mgu(A, A')$ and $A\sigma$ is selected in $(C \vee A \vee A' \llbracket \varphi \rrbracket)\sigma$.

Proposition 11. *Positive Factoring is correct.*

The proof follows directly from the definition of correctness.

Paramodulation

$$\frac{C \vee s \approx t \llbracket \varphi \rrbracket \qquad L[s'] \vee D \llbracket \Psi \rrbracket}{(C \vee L[t] \vee D \llbracket (s \approx t, C, \{C \vee s \approx t\}), (s \not\approx t, D \vee L[t], \{L[s'] \vee D\}), \varphi', \Psi' \rrbracket)\sigma}$$

where $\sigma = mgu(s, s')$, $s\sigma \approx t\sigma$ is selected in $(C \vee s \approx t \llbracket \varphi \rrbracket)\sigma$, $L[s']\sigma$ is selected in $(L[s'] \vee D \llbracket \Psi \rrbracket)\sigma$, $t\sigma \not\succ s\sigma$, and s' is not a variable.

Also, φ' is the same as φ, except that any triple in φ of the form $(L_i, C' \vee s \approx t, S_i)$ is replaced in φ' by the triple $(L_i, C' \vee L[t] \vee D, S_i \cup \{L[s'] \vee D\})$.

Similarly, Ψ' is the same as Ψ, except that any triple in Ψ of the form $(M_i, D' \vee L[s'], T_i)$ is replaced in Ψ' by the triple $(M_i, D' \vee C \vee L[t], T_i \cup \{C \vee s \approx t\})$.

Paramodulation is an inference rule for first order equality that generalizes the substitution of equals by equals. It reduces to Knuth–Bendix completion if all clauses are unit equations.

Proposition 12. *Paramodulation is correct.*

Proof. The premises imply the conclusion, but we must also show that the conclusion is correct. First, we note that $C\sigma \subseteq (C \vee D)\sigma$, $D\sigma \subseteq (C \vee D)\sigma$ and each C_i and D_i has been modified so that $C_i\sigma \subseteq (C \vee D)\sigma$ and $D_i\sigma \subseteq (C \vee D)\sigma$. Therefore the first condition holds.

Also, we see that for all ground substitutions η, $(C \vee s \approx t)\sigma\eta \models (s \approx t)\sigma\eta \vee C\sigma\eta$ and $(L[s'] \vee D)\sigma\eta \models (s \not\approx t)\sigma\eta \vee D\sigma\eta \vee L[t]\sigma\eta$. So trivially, each new local simplifier satisfies the second condition. In addition, we must examine each local simplifier that was modified by the inference. There is an $S' \leq S_i\sigma\eta$ such that $S' \models (C' \vee s \approx t \vee L_i)\sigma\eta$, therefore $S' \cup \{(L[s'] \vee D)\sigma\eta\} \models C'\sigma\eta \vee L[t]\sigma\eta \vee D\sigma\eta \vee L_i\sigma\eta$. Similarly, there is a $T' \leq T_i\sigma\eta$ such that $T' \models (D' \vee L[s'] \vee M_i)\sigma\eta$, therefore $T' \cup \{(C \vee s \approx t)\sigma\eta\} \models D'\sigma\eta \vee C\sigma\eta \vee L[t]\sigma\eta \vee M_i\sigma\eta$. $\square$

In paramodulation inference systems, in order not to lose completeness, we must either allow paramodulation into the smaller side of an equation in some cases, or we must add an inference rule called Equation Factoring. Here we show how the Equation Factoring inference rule would be used in our system.

Equation Factoring

$$\frac{C \vee s \approx t \vee s' \approx t' \llbracket \varphi \rrbracket}{(C \vee t \not\approx t' \vee s' \approx t' \llbracket \varphi' \rrbracket)\sigma}$$

where $\sigma = mgu(s, s')$, $s\sigma \approx t\sigma$ is selected in $(C \vee s \approx t \vee s' \approx t' \llbracket \varphi \rrbracket)\sigma$, and $t\sigma \not\succ s\sigma$.

Also, φ' is the same as φ, except that any triple in φ of the form $(L_i, C' \vee s \approx t, S_i)$ is replaced in φ' by the triple $(L_i, C' \vee t \not\approx t' \vee s' \approx t', S_i)$.

Proposition 13. *Equation Factoring is correct.*

Proof. The premise implies the conclusion and each C_i has been modified so that $C_i\sigma \subseteq (C \vee t \not\approx t' \vee s' \approx t')\sigma$. In addition, for any modified local simplifier and ground substitution η, there is some $S' \leq S_i\sigma\eta$ such that $S' \models (C' \vee s \approx t \vee L_i)\sigma\eta$, therefore $S' \models (C' \vee t \not\approx t' \vee s' \approx t' \vee L_i)\sigma\eta$. $\square$

The last inference rule necessary for completeness is Equation Resolution. It generalizes the notion of removing trivial disequations. An alternative to Equation Resolution is to add the Reflexivity axiom $x \approx x$ to the set of clauses.

Equation Resolution

$$\frac{C \vee s \not\approx t \llbracket \varphi \rrbracket}{(C \llbracket \varphi' \rrbracket)\sigma}$$

where $\sigma = mgu(s, t)$ and $s\sigma \not\approx t\sigma$ is selected in $(C \vee s \not\approx t \llbracket \varphi \rrbracket)\sigma$.

Also, φ' is the same as φ, except that any triple of the form $(L_i, C' \vee s \not\approx t, S_i)$ is replaced by (L_i, C', S_i).

Proposition 14. *Equation Resolution is correct.*

The proof follows directly from the definition of correctness.

4 The Deletion Rules

In this section we present some well-known deletion rules, (e.g., subsumption and demodulation) to indicate how they are affected by the local simplifiers. The reader will note that, unlike other theorem provers with efficiency constraints, the local simplifier on the simplifier and subsumer do not have to be modified in order to perform the deletion. Recall that the difference between an inference rule and a deletion rule is that an inference rule *must* be performed and a deletion rule *may* be performed if desired. We present deletion rules in the form:

$$T \quad \Rightarrow \quad T'$$

where T and T' are sets of clauses.

This means that, if T is a subset of the existing set of clauses, then we may delete all the members of T and add all the members of T'. Generally T and T' have some elements in common. If those common elements exist, then we may delete the members of $T \setminus T'$ as long as we add the members of $T' \setminus T$. Sometimes T will be empty, meaning that the members of T may be deleted without adding anything. As in inference rules, the members of T are called the *premises*, and the members of $T' \setminus T$ are the *conclusions*. A deletion rule may be performed at any time. We will define it so that the deleted clauses are *redundant*.

In addition to redundant clauses, we need to define redundant inferences (Bachmair & Ganzinger (1994)). Redundant inferences do not need to be performed, because they are not needed for the proof.

Definition 15. An inference

$$\frac{C_1 \cdots C_n}{C\sigma}$$

is redundant in S if any of $C_1\sigma, \cdots C_n\sigma$ is redundant or if $C\sigma$ is redundant at $C_n\sigma$ in S.

We define a function called *height* for each clause.

Definition 16. For all initial clauses C, $height(C) = \{C\}$. If a clause $C\sigma$ is created by the inference

$$\frac{C_1 \cdots C_n}{C\sigma}$$

then $height(C\sigma) = \{C_n\sigma\}$. However, as soon as $C\sigma$ has been used in an inference or a deletion rule, then $height(C\sigma) = \{C\sigma\}$.

This function will be used to show redundancy. If a clause C is redundant at $height(C)$, then either C is redundant or the inference used to create C is redundant. In either case, C can be removed.

Now we define what it means for a deletion rule to be correct.

Definition 17. A deletion rule is *correct* if, for all S such that $T \subseteq S$ and each C in T is correct in S,

1. $S \models C$ for all $C \in T'$,
2. C is correct in S for all $C \in T'$, and
3. for each member C of $T \setminus T'$, either C is redundant at $height(C)$ in $S \setminus T \cup T'$ or there is a member D of $S \setminus T \cup T'$ such that $D\sigma = C$ for some σ.

If $T' = \emptyset$, cases 1 and 2 are not necessary and case 3 says that each member of T is redundant in S.

As mentioned above, the clauses deleted by correct deletion rules are redundant or were produced by a redundant inference. In this section, we will present deletion rules which remove redundant clauses. In the section on Local Simplification, we will present deletion rules which remove clauses produced by redundant inferences.

The deletion rules are as follows:

Subsumption

$$\{C[\![\varphi]\!], \quad C\theta \vee D[\![\Psi]\!]\} \quad \Rightarrow \quad \{C[\![\varphi']\!]\}$$

where θ is any substitution, and $\varphi' = \varphi \cup \{(L_i, C_i, S_i) \in \Psi \mid C_i \subseteq C\}$.

Subsumption is interesting because, in constrained theorem proving systems we must remove constraints from φ in order to perform the subsumption, whereas in our case we don't have to remove local simplifiers from φ. In fact, we may actually add local simplifiers to φ.

Proposition 18. *Subsumption is correct.*

Proof. The left premise is smaller than and implies the right premise, therefore the right premise is redundant. But this deletion rule is slightly different from the other deletion rules. In this rule, we modified φ. So we must show that the modification of φ is correct. The only thing that needs checking is that any (L_i, C_i, S_i) added to φ has the property that $C_i \subseteq C$. But that must be true, because that is the condition that allows us to add it. □

Unit Simplification

$$\{L[\![\varphi]\!], \quad \bar{L}\theta \vee D[\![\Psi]\!]\} \quad \Rightarrow \quad \{L[\![\varphi]\!], \quad D[\![\Psi']\!]\}$$

where θ is any substitution.

Ψ' is the same as Ψ, except that every local simplifier of the form $(L_i, \bar{L}\theta \vee C', S_i)$ is replaced by $(L_i, C', S_i \cup \{L\})$.

Proposition 19. *Unit Simplification is correct.*

Proof. The conclusion is implied by the premises. The right premise is redundant in the presence of the left premise and the conclusion. We must show that the conclusion is correct. Since the right premise is correct, we know that $C' \subseteq D$. Also, for any modified local simplifier and ground substitution η, there exists an $S' \leq S_i\eta$ such that $S' \models (\bar{L}\theta \vee C' \vee L_i)\eta$, therefore $S' \cup \{L\eta\} \models (C' \vee L_i)\eta$. □

Demodulation

$$\{s \approx t\,[\![\varphi]\!],\quad L[s\theta] \vee D\,[\![\Psi]\!]\} \quad\Rightarrow\quad \{s \approx t\,[\![\varphi]\!],\quad L[t\theta] \vee D\,[\![\Psi']\!]\}$$

for any substitution θ, if $s\theta > t\theta$.

The local simplifier Ψ' is the same as Ψ, except that every occurrence of $(L_i, L[s\theta] \vee C', S_i)$ is replaced by $(L_i, L[t\theta] \vee C', S_i \cup \{s \approx t\})$

Proposition 20. *Demodulation is correct.*

Proof. The conclusion is implied by the premises. The right premise is redundant in the presence of the left premise and the conclusion. We must show that the conclusion is correct. Since the right premise is correct, we know that $C' \vee L[t\theta] \subseteq D$. For any modified local simplifier and ground substitution η, there exists an $S' \leq S_i\eta$ such that $S' \models (L[s\theta] \vee C' \vee L_i)\eta$, therefore $S' \cup \{s\eta \approx t\eta\} \models (L[t\theta] \vee C' \vee L_i)\eta$. $\square$

We now define other deletion rules, which are often not presented in theoretical papers because they follow from the definition of redundancy. However, these are rules which are often used in practice. We present them here to make explicit the way that local simplifiers are handled. Each of the correctness proofs follows immediately from the definition of correctness.

Tautology Deletion

$$\{C \vee A \vee \neg A\,[\![\varphi]\!]\} \quad\Rightarrow\quad \emptyset$$

Proposition 21. *Tautology Deletion is a correct deletion rule.*

This is trivially redundant, i.e., $n = 0$ in the definition of redundancy.

Equational Tautology Deletion

$$\{C \vee s \approx s\,[\![\varphi]\!]\} \quad\Rightarrow\quad \emptyset$$

Proposition 22. *Equational Tautology Deletion is correct.*

Thinning

$$\{C \vee L \vee L\,[\![\varphi]\!]\} \quad\Rightarrow\quad \{C \vee L\,[\![\varphi]\!]\}$$

Proposition 23. *Thinning is correct.*

Equational Thinning

$$\{C \vee s \not\approx s\,[\![\varphi]\!]\} \quad\Rightarrow\quad \{C\,[\![\varphi']\!]\}$$

The local simplifier φ' is the same as φ, except that any triple of the form $(L_i, C' \vee s \not\approx s, S_i)$ is replaced by (L_i, C', S_i).

Proposition 24. *Equational Thinning is correct.*

Proof. The premise implies the conclusion, and the conclusion is smaller than and implies the premise. To see that the conclusion is correct, we note that any S' that implies some instance of $C' \vee s \not\approx s \vee L_i$ must also imply the same instance of $C' \vee L_i$. $\square$

5 Local Simplification

In the previous sections we showed how the local simplifiers are generated and how they are inherited from their ancestors. In this section, we show the significant result of this paper. We show how the local simplifiers can be used to perform simplifications on their associated clauses. The rules in this section are called local simplification rules. The first rule, called *Local Subsumption* shows how the local simplifiers of a clause can be used to delete that clause. The local simplification rules *Local Unit Simplification*, *Local Demodulation* and *Self Demodulation* limit the search space by simplifying a clause, and also may shorten the length of the proof. We will have a strong and weak version of each local simplification rule. The strong version allows us to delete a clause, while the weak version does not.

Local simplification rules are deletion rules, which may be applied to a clause immediately after that clause is formed by an inference. This is the reason we have defined the function *height*. In practice it makes sense to simplify a clause immediately after it is created. Therefore, as defined in the section on Deletion Rules, when a clause C is created, $height(C)$ is set to detect if the inference creating C was redundant. However, an implementation may decide to perform local simplification later. In that case, if a clause has been used in an inference or another deletion rule, $height(C)$ is set to detect that C is redundant.

A Local Simplification rule is a special deletion rule that uses the information stored with each clause to detect if the inference producing a clause is redundant. Therefore, it is defined as a deletion rule. We call it a *strong local simplification* if it is a correct deletion rule. We call it a *weak local simplification* if it satisfies properties 1 and 2 of the definition of a correct deletion rule. In the case of a weak local simplification, it is possible to add the new clauses which are generated, but not to delete the old clauses. This could potentially be useful, because there are some cases when a clause can be simplified even though the original clause may not be deleted.

The first local simplification we present is called local subsumption, because we show that a clause may be removed since a subset of it is implied by smaller clauses. [4]

Local Subsumption

$$\{C \vee L\,[\![\,\varphi[(L', C_i, S_i)]\,]\!]\} \quad \Rightarrow \quad \emptyset$$

where $\theta = mgu(L, L')$, $(C \vee L)\theta = C \vee L$ [5], and $S_i\theta < height(C \vee L\,[\![\,\varphi\,]\!])$. Recall that $height(C \vee L\,[\![\,\varphi\,]\!])$ is the right premise of the inference which produced $C \vee L$, when $C \vee L$ is created. If local subsumption is performed after $C \vee L$ is used in an inference or a deletion rule, then the height is the clause itself. This is a strong local simplification.

---

[4] The notation $\varphi[(L', C_i, S_i)]$ means that φ contains the local simplifier (L', C_i, S_i).

[5] A special case of this is when $L = L'$, though it is not the only case.

Proposition 25. *Local Subsumption is correct.*

Proof. We assume that $C \vee L [\![\varphi[(L', C_i, S_i)]]\!]$ is correct. We must show that it is redundant at $height(C \vee L [\![\varphi]\!])$ in S. Since it is correct, we know that for all ground substitutions η, there exists an S' such that $S' \leq S_i\theta\eta$ and $S' \models L\theta\eta \vee C_i\theta\eta$. Since $C_i\theta\eta \subseteq C\theta\eta \vee L\theta\eta$, we know $S' \models C\theta\eta \vee L\theta\eta = C\eta \vee L\eta$. And $S' \leq S_i\theta\eta < height(C \vee L [\![\varphi]\!])\eta$. So $C \vee L$ is redundant at $height(C \vee L [\![\varphi]\!])$ in S. $\square$

Note that it is not enough that the ancestor literal in the local simplifier matches onto the literal in the clause. For instance, if we have the clause $q(x) \vee p(a) [\![(p(x), q(x), S_i)]\!]$ then for all ground substitutions η there exists an $S' \leq S_i\eta$ such that $S' \models (p(x) \vee q(x))\eta$ but it is not necessarily the case that $S' \models (q(x) \vee p(a))\eta$. Similarly, given the clause $q(a) \vee p(x) [\![(p(b), q(a), S_i)]\!]$ then there exists an S' such that $S' \models p(b) \vee q(a)$ but it is not necessarily the case that for every ground substitution η, $S' \models (q(a) \vee p(x))\eta$. So it is also not enough to say that the literal in the clause matches onto the ancestor literal in the local simplifier.

In *local unit simplification*, one of the literals in the clause may be deleted.

Local Unit Simplification

$$\{C \vee \bar{L} [\![\varphi[(L', C_i, S_i)]]\!]\} \quad \Rightarrow \quad \{(C [\![\varphi']\!])\theta\}$$

where $\theta = mgu(L, L')$ and $\bar{L} \notin C_i$. This is the weak version of local unit simplification. The strong version is when, in addition, $C\theta = C$.

The local simplifier φ' is the same as φ, except that any local simplifier of the form $(L_j, C' \vee \bar{L}, S_j)$ is replaced by $(L_j, C' \vee C_i, S_j \cup S_i)$.

Proposition 26. *Local Unit Simplification is correct.*

Proof. We assume that $C \vee \bar{L} [\![\varphi[(L', C_i, S_i)]]\!]$ is correct. We must show that $(C [\![\varphi']\!])\theta$ is correct. We see that $C' \vee C_i \subseteq C$, because $\bar{L} \notin C_i$. For every ground substitution η, there exists an $S' \leq S_i\theta\eta$ such that $S' \models C_i\theta\eta \vee L'\theta\eta$. Also, there exists an $S'' \leq S_j\theta\eta$ such that $S'' \models C'\theta\eta \vee \bar{L}\theta\eta \vee L_j\theta\eta$. Therefore, $S' \cup S'' \models C_i\theta\eta \vee C'\theta\eta \vee L_j\theta\eta$.

To prove condition 2, $S' \cup \{C\theta\eta \vee \bar{L}\theta\eta\} \models C\theta\eta$ since $C_i \subseteq C$. So the weak version of local unit simplification is correct.

For the strong version, $C\theta = C \models C \vee \bar{L}$, therefore $C \vee \bar{L}$ is redundant at $height(C \vee \bar{L} [\![\varphi]\!])$ in $S \cup \{C\theta\}$. $\square$

As an example of the use of local unit simplification, consider the clauses $p \vee q$, $\neg q \vee r$ and $\neg r \vee \neg q$. If we resolve the first two clauses, we get $p \vee r [\![(q, p, \{p \vee q\}), (\neg q, r, \{\neg q \vee r\})]\!]$. This can now be resolved with $\neg r \vee \neg q$ resulting in $p \vee \neg q [\![(q, p, \{p \vee q\}), (\neg q, \square, \{\neg q \vee r, \neg r \vee \neg q\}), (r, p, \{p \vee r\}), (\neg r, \neg q, \{\neg q \vee \neg r\})]\!]$. Note that by definition 7, the local simplifier $(\neg q, \neg q, \{\neg q \vee r, \neg r \vee \neg q\}$ should have appeared in the clause. However, by the assumption following definition 7, the local simplifier was modified to $(\neg q, \Box, \{\neg q \vee r, \neg r \vee \neg q\}$. We can perform a unit simplification to remove the literal $\neg q$ resulting in the clause p.[6]

To show why we need the condition that $\bar{L} \notin C_i$, consider the following example. Suppose we are given the clauses $p \vee q$, $\neg q \vee r$ and $\neg p \vee \neg q$, with $q > p > r$. If we resolve the first two clauses, we get $p \vee r \llbracket (q, p, \{p \vee q\}), (\neg q, r, \{\neg q \vee r\}) \rrbracket$. This can now be resolved with $\neg p \vee \neg q$ resulting in $\neg q \vee r \llbracket (q, \neg q, \{p \vee q, \neg p \vee \neg q\}), (\neg q, r, \{\neg q \vee r\}), (p, r, \{p \vee r\}), (\neg p, \neg q, \{\neg p \vee \neg q\}) \rrbracket$. If we ignored the condition, we could perform a unit simplification to remove the literal $\neg q$ resulting in the clause r with some local simplifiers. This does not follow from the original clauses. However, the second local simplifier allows us to perform a local subsumption to remove the clause entirely. Of course, we could remove the clause anyway, because it already exists.

For the strong version of Local Unit Simplification, it is not enough that the ancestor literal in the local simplifier matches onto the literal in the clause. For instance, if we have the clause $q(x) \vee \neg p(a) \llbracket (p(x), q(x), S_i) \rrbracket$ then for all ground substitutions η, there exists an $S' \leq S_i\eta$ such that $S' \models (p(x) \vee q(x))\eta$ therefore $S' \cup \{q(x) \vee \neg p(a)\} \models q(a)$. So we are allowed to add the clause $q(a)$ but we may not delete $q(x) \vee p(a)$. However, given the clause $q(a) \vee \neg p(x) \llbracket (p(b), q(a), S_i) \rrbracket$ then for every ground substitution η there exists an $S' \leq S_i\eta$ such that $S' \cup \{q(a) \vee \neg p(x)\} \models p(b) \vee q(a)$ but it is not necessarily the case that $S' \models q(a)$.

Now we present two forms of local demodulation. The first version is similar to local unit simplification, except that it deals with equalities.

Local Demodulation

$$\{C \vee L[s'] \llbracket \varphi[(s \approx t, C_i, S_i)] \rrbracket\} \quad \Rightarrow \quad \{(C \vee L[t] \llbracket \varphi' \rrbracket)\theta\}$$

where $\theta = mgu(s, s')$ and $L[s'] \notin C_i$. This is the weak version. The strong version is where, in addition, $S_i\theta < height(C \vee L[s'] \llbracket \varphi \rrbracket)$, $s\theta > t\theta$, and $(C \vee L[s'])\theta = C \vee L[s']$.

The local simplifier φ' is the same as φ, except that any local simplifier of the form $(s \not\approx t, C' \vee L[s'], S_j)$ is replaced by $(s \not\approx t, C' \vee L[t], S_j)$, and any local simplifier of the form $(L_j, C' \vee L[s'], S_j)$ where $L_j \neq (s \not\approx t)$ is replaced by $(L_j, C' \vee C_i \vee L[t], S_j \cup S_i)$.

Proposition 27. *Local Demodulation is correct.*

Proof. We assume that $C \vee L[s'] \llbracket \varphi[(s \approx t, C_i, S_i)] \rrbracket$ is correct. We must show that $(C \vee L[t] \llbracket \varphi' \rrbracket)\theta$ is correct. For the case where L_j is different from $s \not\approx t$, we see that $C' \vee C_i \vee L[t] \subseteq C \vee L[t]$ because $L[s'] \notin C_i$. In addition, for every ground substitution η, there exists an $S' \leq S_i\theta\eta$ such that $S' \models (C_i \vee s \approx t)\theta\eta$. Also, there exists an $S'' \leq S_j\theta\eta$ such that $S'' \models (C' \vee L[s'] \vee L_j)\theta\eta$. Therefore, $S' \cup S'' \models (C_i \vee C' \vee L[t] \vee L_j)\theta\eta$.

The case where $s \not\approx t$ is simpler. In that case, $S'' \models (s \not\approx t \vee C' \vee L[s'])\theta\eta$. Therefore $S'' \models (s \not\approx t \vee C' \vee L[t])\theta\eta$.

[6] We will see in section 8 that the local simplifiers on unit clauses can be removed.

For condition 2, $S' \cup \{C\theta\eta \vee L[s']\theta\eta\} \models C\theta\eta \vee L[t]\theta\eta$ since $C_i \subseteq C$. This proves the correctness of weak local demodulation.

Now we prove the strong version. Since $S' \models (C_i \vee s \approx t)\theta\eta$ and $C_i\theta\eta \subseteq C\theta\eta$, then $S' \cup \{(C \vee L[t])\theta\eta\} \models (C \vee L[s'])\theta\eta = (C \vee L[s'])\eta$. And since $S' \leq S_i\theta\eta < height(C \vee L[s'] \llbracket \varphi \rrbracket)\eta$, $C \vee L[s]$ is redundant at $height(C \vee L[s'] \llbracket \varphi \rrbracket)$ in $S \cup \{(C \vee L[t])\theta\}$. $\square$

There is a generalization of the paramodulation rule called parallel paramodulation. In that rule, whenever $C \vee s \approx t$ is paramodulated with $L[s'] \vee D$, every position where s' is a subterm of $L[s'] \vee D$, is replaced by t. This cuts down the search space, especially if there are lots of clauses involving equations with a left hand side of s. We can simulate that inference rule, because paramodulation would save $s \approx t$ as an ancestor literal in a local simplifier and local demodulation would be used to apply it to all the other positions in which s' appears.

We give an example of local demodulation. Suppose we have an ordering where all equality predicates are smaller than all non-equality predicates. Consider the paramodulation of $c \approx d \vee a \approx b$ and $P(a) \vee Q(c)$. The result is $c \approx d \vee P(b) \vee Q(c) \llbracket \varphi_1 \rrbracket$, where one of the local simplifiers in φ_1 is $(a \approx b, c \approx d, \{a \approx b \vee c \approx d\})$. If we then resolve this clause with $\neg Q(c) \vee R(a)$, we get $c \approx d \vee P(b) \vee R(a) \llbracket \varphi_2 \rrbracket$, and φ_2 also contains the local simplifier $(a \approx b, c \approx d, \{a \approx b \vee c \approx d\})$. Then we can apply the local demodulation rule to simplify $c \approx d \vee P(b) \vee R(a)$ to $c \approx d \vee P(b) \vee R(b)$. Supposing that we did not have the local demodulation rule. If $\neg Q(c)$ was selected, we could have instead performed a paramodulation among the clause $c \approx d \vee a \approx b$ to result in a clause which factored out to the same clause. But we would not have known that we could delete the clause $c \approx d \vee P(b) \vee R(a)$, and if the equation $c \approx d$ had disappeared by a later inference, we couldn't have even done the factoring. That illustrates the benefit of local demodulation.

For the same reasons as in local unit simplification, in order to perform strong local demodulation we needed to require that $(C \vee L[s'])\theta = C \vee L[s']$. We also needed that $S_i < height(C \vee L[s'] \llbracket \varphi \rrbracket)$ to preserve completeness. This is so we can guarantee that a clause is being simplified by smaller clauses.

The next version of the local simplification rule does not use the local simplifier for simplification. Instead, it uses a negative equality in the clause. The strong version of this rule is originally from Boyer & Moore (1979) and called *contextual rewriting*. It does not use the local simplifiers, but we present it here because it fits neatly into our framework.

Self Demodulation

$$\{s \not\approx t \vee C \vee L[s'] \llbracket \varphi \rrbracket\} \quad \Rightarrow \quad \{(s \not\approx t \vee C \vee L[t] \llbracket \varphi' \rrbracket)\theta\}$$

This is the weak version. The strong version is where θ is the identity and $s > t$.

The local simplifier φ' is just like φ, except that any local simplifier of the form $(L_j, C' \vee L[s'], S_j)$ is replaced by $(L_j, C' \vee L[t] \vee s \not\approx t, S_j)$.

Proposition 28. *Self Demodulation is correct.*

Proof. We assume that $s \not\approx t \vee C \vee L[s'] [\![\varphi]\!]$ is correct. We must show that $(s \not\approx t \vee C \vee L[t] [\![\varphi']\!])\theta$ is correct. Trivially, $C' \vee L[t] \vee s \not\approx t \subseteq s \not\approx t \vee C \vee L[t]$. In addition, for all ground substitutions η, there exists an $S' \leq S_j\theta\eta$ such that $S' \models (C' \vee L[s'] \vee L_j)\theta\eta$. Therefore, $S' \models (C' \vee L[t] \vee s \not\approx t \vee L_j)\theta\eta$. For condition 2, $(s \not\approx t \vee C \vee L[s'])\theta\eta \models (s \not\approx t \vee C \vee L[t])\theta\eta$. For the strong version, $(s \not\approx t \vee C \vee L[t])\theta\eta \models (s \not\approx t \vee C \vee L[s])\eta$, so $s \not\approx t \vee C \vee L[s]$ is redundant at $height(s \not\approx t \vee C \vee L[s'] [\![\varphi]\!])$ in $S \cup \{(s \not\approx t \vee C \vee L[t])\theta\}$. $\square$

6 Completeness

In this section we prove the completeness of the inference system given in this paper. The definitions of correctness imply that the inference system is sound. To prove completeness, we use results from Bachmair & Ganzinger (1994) and Bachmair et al (1995).

From Bachmair & Ganzinger (1994), we get the definition of a fair theorem proving derivation, meant to model an automated theorem prover.

Definition 29. A (finite or countably infinite) sequence $S_0, S_1, S_2, \ldots$ of sets of clauses is called a *theorem proving derivation* if each set S_{i+1} can be obtained from S_i by adding a clause which is a consequence of S_i or by deletion of a redundant or subsumed clause in S_i. A clause C is said to be *persisting* if there exists some j such that for every $k \geq j$, $C \in S_k$. The set of all persisting clauses, denoted S_∞, is called the *limit* of the derivation.

Let Sel be a selection rule. Let I_{Sel} be the inference rules in this paper instantiated by Sel. A set of clauses S is *Sel-saturated* if every inference from I_{Sel} applied to clauses in S is redundant in S. A theorem proving derivation is called *Sel-fair* if S_∞ is Sel-saturated. A derivation is called *fair* if it is Sel-fair for some valid Sel.

The set of inference rules proved complete in Bachmair & Ganzinger (1994) and Bachmair et al (1995) is just I_{Sel} for a valid Sel, ignoring the local simplifiers. Therefore, the completeness results from those papers can be used. This illustrates the beauty of that abstract definition of an inference system. We do not need to re–prove their completeness proof. We only need to apply it to our situation. The way it has been applied to our situation is to show that the local simplification rules remove redundant clauses, which we have done by showing the correctness of those rules.

Theorem 30. *Let $S_0, S_1, S_2, \ldots$ be an fair theorem proving derivation. If S_0 is unsatisfiable then $\square \in S_\infty$.*

See Bachmair & Ganzinger (1994) and Bachmair et al (1995) for the proof of the theorem. This says that any fair theorem proving derivation will produce the empty clause. We must show that this completeness result applies to the local simplification inference system. Let D be the set of deletion rules in this paper and L be the set of local simplification rules. Then the following theorem immediately follows from the correctness of the local simplification rules.

Theorem 31. *Let S_0 be a set of (undecorated) clauses. Consider the sequence $S_0, S_1, S_2, \ldots$ where each S_{i+1} is obtained from S_i by applying a rule from I, L, or D to clauses from S_i. If the sequence is fair, then $\Box \in S_\infty$ if and only if S_0 is unsatisfiable.*

Proof. Since the inference rules I are the same as in Bachmair and Ganzinger (1994) and Bachmair et al (1995), the proof follows from the correctness of the inference and deletion rules. By the correctness of these rules, each clause that is created is correct. Therefore, each clause that is added, logically follows from the existing clauses. Also, by definition of correct deletion rule, each clause that is removed is redundant or the inference that has just created it is redundant. So the proof follows directly from the previous theorem. $\Box$

7 Complexity of SLD Resolution

When restrictions on inference rules are proposed, it is generally not shown theoretically that the restriction makes theorem proving more efficient. Ideally, it would be helpful to show that a restriction reduces the search space. We cannot show that theoretically in this case. But we can show theoretically that Local Simplification reduces the sizes of the proofs in some settings. In particular, we show that for ground horn clauses, with any selection rule, SLD resolution has a polynomial proof (this yields a variant of an algorithm from Dowling & Gallier (1984). However, without local simplification, we exhibit a set of clauses and a selection rule so that SLD resolution only has exponential proofs.

A horn clause is a clause with at most one positive literal. A clause with exactly one positive literal is called a *program clause* and a clause with no positive literals is called a *goal clause*. SLD resolution is an inference system containing the rules given in this paper applied to horn clauses, except that the selection rule is modified so that the positive literal is selected in each program clause and a negative literal is selected in each goal clause. Notice that the resolution inference rule is the only one that applies to horn clauses. SLD resolution has been proved complete for horn clauses. The SLD resolution selection rule cannot be encoded by the selection rule in this paper. Therefore, to discuss local simplification in SLD resolution, we must show how to use the selection rule in this paper to show completeness of SLD resolution. First we define some sets used to give an ordering.

Definition 32. Given a set S of clauses, we define a hierarchy of sets of positive literals. Let $M_0 = \{A | A \in Gr(S)\}$. For $n \geq 0$, $M_{n+1} = \{A | \neg B_1 \vee \cdots \vee \neg B_m \vee A \in Gr(S)$ and $\{B_1, \cdots, B_m\} \subseteq M_n\}$. Let $M_\infty = \bigcup_{n \geq 0} M_n$. If a literal A is in M_0, we say that A is at level 0. If A is in M_{n+1} but not in M_n, we say that A is at level $n+1$. Now we define a total ordering $\succ$ on the ground literals with the property that for all positive ground literals A and B, $A \succ B$ if

- A is at level i, B is at level j and $i > j$ or
- $A \notin M_\infty$ and $B \in M_\infty$.

We have not specified how to compare A and B if they are both at level i for some i, or if neither are in M_∞. In those cases we do not care. The ordering is extended to negative literals so that, for all i, $\neg A \succ B$ for all literals A, B at level i. Also $A \succ B$ if and only if $\neg A \succ \neg B$.

Using this ordering, we can prove some facts about ground horn clauses. First we define the following sets given a set S of clauses from which $M_0, M_1, M_2, \cdots$ have been defined. Let $T_0 = \{C \mid \exists A \in C \text{ such that } A \notin M_\infty\}$. Let $T_1 = \{C \mid \exists \neg A \in C \text{ such that } A \notin M_\infty\}$. $T_2 = \{C \mid \exists \neg A, B \in C \text{ such that } A \text{ is at level } i, B \text{ is at level } j \text{ and } i \geq j\}$.

Proposition 33. *Let C be a ground horn clause in S. If $C \in T_2$ then C is redundant in S.*

Proof. Let C be of the form $\neg A \vee B \vee \Gamma$ where A is at level i, B is at level j and $i \geq j$. Then there must be a set of clauses $\{C_1, \cdots, C_n\} \subseteq S$ which implies B such that each C_k is smaller than C. Therefore C is redundant in S. □

Proposition 34. *Let C_1 and C_2 be ground horn clauses in S. If $C_1 \in T_0 \cup T_1$ or $C_2 \in T_0 \cup T_1$ and D is the conclusion of a resolution inference among C_1 and C_2, then $D \in T_0 \cup T_1$.*

Proof. Let $C_1 = \Gamma \vee A$ and $C_2 = \neg A \vee \Delta$ such that C_1 or C_2 is in $T_0 \cup T_1$. If $A \in M_\infty$ then there exists some literal B or $\neg B$ in Γ or Δ such that B is not in M_∞. Therefore $D \in T_0 \cup T_1$. If $A \notin M_\infty$ then there exists some literal $\neg B$ in Γ such that B is not in M_∞ so $D \in T_0 \cup T_1$. □

We define a selection rule to be an *SLD selection rule* if it selects a negative literal in each goal clause and the positive literal in each program clause. We define a selection rule to be a *Max selection rule* if it selects a negative literal in each goal clause and all maximal literals in each program clauses. An inference using an SLD selection rule is called an *SLD inference*. An inference using a Max selection rule is called a *Max inference*. Two selection rules are called *compatible* if they select the same literal in each goal clause. We have the following useful proposition.

Proposition 35. *Let Sel_{SLD} be an SLD selection rule and Sel_{Max} be a Max selection rule, such that Sel_{SLD} is compatible with Sel_{Max}. If $C \notin T_0 \cup T_1 \cup T_2$, then Sel_{SLD} and Sel_{Max} select the same literal in C.*

Proof. If C is a goal clause, then the proposition is true by compatibility. If C is a program clause not in $T_0 \cup T_1 \cup T_2$ then C is of the form $\Gamma \vee A$ where A is at level j for some j and each $B \in \Gamma$ is at level i for some i such that $i < j$. Therefore, A is maximal in C. So Sel_{Max} and Sel_{SLD} both select A. □

Now we can prove the completeness of SLD resolution using the completeness of Max resolution. This will show that local simplification can be performed in SLD resolution without losing completeness.

Theorem 36. *Let Sel be an SLD selection rule. Let S be a set of unsatisfiable horn clauses. Let $S_0, S_1, S_2, \cdots$ be a Sel-fair theorem proving derivation from S. Then $\Box \in S_\infty$.*

Proof. Let S^* be the set of clauses formed by closing S_∞ under non–redundant Max inferences for some Max selection rule compatible with Sel. By proposition 35, all of these Max inferences involve some clause in $T_0 \cup T_1 \cup T_2$. Otherwise, they would have already been performed and would now be redundant. By proposition 33, none of the Max inferences involve clauses from T_2. Therefore, all the Max inferences involve some clause from $T_0 \cup T_1$. By completeness, we know that $\Box \in S^*$. By proposition 34, the proof of $\Box$ does not involve clauses from $T_0 \cup T_1$. Therefore, $\Box \in S_\infty$. $\Box$

This shows the completeness of SLD resolution. These results are similar to results from Bachmair & Ganzinger (1991). It shows that all the notions of redundancy apply to SLD resolution. Therefore local simplification applies. Now we show that any set of ground horn clauses has a polynomial local simplification proof under any SLD selection rule. Then we will show this is not the case without local simplification.

Theorem 37. *Let S be an unsatisfiable ground set of clauses and Sel be an SLD selection rule. Let n be the number of distinct positive literals in S. Then S has a proof, using selection rule Sel, containing at most $n+1$ clauses.*

Proof. We may assume that S contains no clauses from $T_0 \cup T_1 \cup T_2$ and only one goal clause. Otherwise, we may remove clauses to make it so without disturbing unsatisfiability. In addition, we assume a proof where local unit simplification is applied whenever it is applicable. Each inference performed is a resolvent among a goal clause C and a program clause of the form $A \vee \Gamma$. This generates a new goal clause D which inherits all the local simplifiers of C with modifications and adds one new local simplifier of the form $(A, \Gamma, \{A \vee \Gamma\})$ to the new goal clause. Since $A \notin T_0 \cup T_1 \cup T_2$, we know that $B \prec A$ for all $B \in \Gamma$. For the same reason, further inferences can only make the elements of Γ smaller. Therefore, every local simplifier whose ancestor literal is positive will be of the form (A, Γ, S) where every element of Γ is smaller than A. That means that whenever A appears in a clause with local simplifier (A, Γ, S), A will be immediately removed from the clause since $A \notin \Gamma$. Therefore it will only be possible in the proof to perform one inference containing A as a positive literal. Which implies that the length of the proof will be the number of positive literals removed plus one for the original clause. $\Box$

However, without local simplification rules, there are some sets of ground horn clauses and SLD selection rules which only have exponential proofs. Consider the following set of clauses.

$$\neg p_1$$
$$p_1 \vee \neg p_2 \vee \neg p_3$$
$$p_2 \vee \neg p_3 \vee \neg p_4$$
$$\vdots$$
$$p_{n-2} \vee \neg p_{n-1} \vee \neg p_n$$
$$p_{n-1} \vee \neg p_n$$
$$p_n$$

Suppose that the $\neg p_i$ with the largest value of i is selected in each goal clause. Then each clause with a positive p_i will be used $fib(i)$ times in any proof, where $fib(i)$ is the i^{th} Fibonacci number. Therefore the proof length is exponential in n. There are some selection rules which have linear proofs for this set of clauses but only if negative factoring is allowed. However, it is impossible to know what is a good selection rule in advance, and negative factoring is rarely used in SLD resolution. Of course other resolution refutations always give linear proofs for horn clauses, but they are not goal-directed.

8 Simplifying Local Simplifiers

Now we show some useful local simplifier simplifications, which either remove local simplifiers that are no longer useful, or move local simplifiers to a place where they will be more useful.

Suppose we have a clause of the form $C \llbracket (L, \bar{L} \vee C', S), \varphi \rrbracket$. The local simplifier $(L, \bar{L} \vee C', S)$ cannot be used for any of the local simplification rules, so we remove the local simplifier and convert the clause to $C \llbracket \varphi \rrbracket$.

If a clause is of the form $C \llbracket (L, C, S), \varphi \rrbracket$, then the local simplifier (L, C, S) is also useless, because since L depends on the whole clause it can never be used for any local simplifications. So we remove the local simplifier and convert the clause to $C \llbracket \varphi \rrbracket$.

The above local simplifier modifications only remove information that is no longer necessary. However, the following local simplifier simplification allows us to use local information globally. Suppose we have a local simplifier of the form $C \llbracket (L, \Box, S), \varphi \rrbracket$. This says that L is true by equations less than or equal to S and depends on nothing in the clause. A local simplifier of this form could arise, because a local simplifier of the form (L, L, S) is converted into $(L, \Box, S)$. Since L depends on nothing in C, it may be useful for simplifying other clauses. So we may remove it from this clause and add it to a list of global simplifiers. Then $(L, \Box, S)$ may be treated as if it was a local simplifier in every clause.

For a particular example, consider a unit clause. Every local simplifier in a unit clause either depends on the entire clause or nothing. In the first case, the local simplifier is removed. In the second case, the local simplifier is moved to a set of global simplifiers.

We note that in every case it is possible to take a local simplifier (L, C, S) and use it as a global simplifier. However, to apply it to a clause D, it is necessary that $C \subseteq D$, which is not likely to be the case. Therefore, we think that in practice it only makes sense to make it a global simplifier when $C = \Box$, or possibly when C contains only one literal. Otherwise, the storage of it might not be worth the limited number of times it is used.

9 Conclusion

We have shown the completeness of the local simplification inference system in combination with a selection rule. Instead of a selection rule, most theorem provers use other kinds of restrictions on the inference rules, like ordered resolution, hyperresolution, set of support resolution, semantic resolution, SLD resolution, ordered paramodulation, strict superposition and hyperparamodulation. All of these restrictions can be simulated by the selection rule we give. In the next few paragraphs, we show how these inference rules can be proved complete by encoding them with valid selection rules. Therefore the local simplification rules apply. We discuss how the local simplification rules affect the efficiency of each selection rule.

Ordered resolution is the resolution inference system where all maximal literals are selected in each clause. Local simplification has a limited benefit for ordered resolution. For a ground clause $C\llbracket (L_i, C_i, S_i), \varphi \rrbracket$ formed by ordered resolution, all literals in C_i are less than L_i and $C_i < S_i$, so the local simplification rules have no benefit. will never be able to be used This illustrates the reason that ordered resolution is an efficient inference rule. It prevents the need to resolve a literal in a clause which was already resolved and removed by an ancestor of the clause. Strict superposition is the paramodulation inference system where all maximal literals are selected in each clause. The above property for ordered resolution is no longer true in this settings, so the local simplifications rules are useful there. Therefore, we believe that the greatest benefit of local simplification is in the paramodulation setting.

Local simplification has more of a benefit when combined with the other resolution restrictions. Many of them are goal directed, so they cannot have the property mentioned of ordered resolution above. Local simplification can be seen as a way of partially retrieving that property.

Hyperresolution is another useful restriction for first order logic. It is a resolution inference system which resolves all the negative literals in a clause at the same time, by positive clauses (clauses containing only positive literals). The conclusion of a hyperresolution inference is a positive clause. Hyperresolution can be simulated by a selection rule which always selects some negative literal if a clause has one. This generates intermediate clauses, but each intermediate clause has a negative literal selected so it cannot be used for anything else but an intermediary of a hyperresolution inference. The local simplification rules are helpful in combination with hyperresolution. Hyperparamodulation is the inference system where all negative literals in a clause are paramodulated into by positive clauses until another positive clause results. This can be viewed as one inference as in hyperresolution. Or it can be encoded like hyperresolution, where a negative literal is selected whenever possible and lots of intermediate clauses are added which can only serve as intermediate clauses.

Semantic resolution is an inference rule where a model I is given and no two clauses are resolved if they are made true by I. In set of support resolution, no two initial clauses are resolved if they are made true by I. These two inference rules can be shown equivalent to hyperresolution under an appropriate mapping from literals true in I to negative literals. The mapping is the following. Consider a set S of ground clauses and I a model. Let S' be a new set of clauses, except that each literal L in S which is made true by I is mapped to a literal $\neg A_L$ in S' and the complement of L is mapped to A_L. Then S is satisfiable if and only if S' is satisfiable and hyperresolution from S derives the empty clause if and only if semantic resolution from S' derives the empty clause. This proves that semantic resolution is complete in the ground case. Therefore, by a standard lifting argument, semantic resolution is complete in general. This same argument proves that set of support resolution is complete. These arguments do not work for equality because paramodulation inferences are not preserved by the translation. In fact, set of support paramodulation is not complete.

In section 7, we showed how to simulate SLD resolution. We also showed that for every selection rule, every ground set of unsatisfiable horn clauses has an SLD refutation (with local simplification) that is linear in the number of literals. However, we exhibit an unsatisfiable set of horn clauses and a selection rule where every SLD refutation (without local simplification) is exponential in the number of literals. Other resolution proof strategies always have linear refutations for horn clauses, but they are not goal directed.

Model elimination theorem provers (Loveland (1968), Loveland (1978)) also employ the strategy of saving literals involved in a resolution, so that they can be used to simplify the clause. In model elimination, there are two kinds of literals: B-literals which are the literals in the clause and A-literals (ancestor literals) which are the literals we put in local simplifiers. In model elimination, the literal resolved against is stored as an A-literal. The position of the A-literal in the clause tells which literals it can simplify. In local simplification, we use the second parameter of the local simplifier to tell which literals can be simplified. The reduction rule of model elimination is similar to the weak unit simplification rule given in this paper. We show how to do this in a resolution framework and show that local simplification is compatible with restrictions and deletion rules. It is our strong local simplification rules that exploit redundancy criteria and allow clauses to be removed. We show local simplification in a paramodulation setting, combined with the other known restrictions of paramodulation and deletion rules. The reduction rule from Model elimination with Paramodulation is similar to weak local demodulation, however function reflexivity and paramodulation into variables is required for completeness and it is not compatible with deletion rules or rewrite techniques (Loveland (1978)).

In this paper, we take advantage of the powerful redundancy criteria of Bachmair & Ganzinger (1994). We save equations as ancestor when each clause is created. We show how to pass that information to the descendants as more inferences are performed. Then we show how to use this information to simplify clauses. The intent is to gain some of the power that simplification provides in completion procedures. Other recent work shows how constraints can reduce the search space of theorem proving procedures (see Bachmair et al (1995), Lynch & Snyder (1995), Nieuwenhuis & Rubio (1995), Nieuwenhuis & Rubio (1992), Nieuwenhuis & Rubio (1994), Lynch & Snyder (1994), Vigneron (1994)). The results of this paper have a nice feature which is not exhibited in constraint theorem provers. That is the fact that we don't need to weaken the local simplifiers in order to perform simplification or subsumption or any other deletion rule. If we were to combine local simplification with basic paramodulation, we would need to weaken constraints to perform deletion rules. However, we believe local simplifiers would be useful in combination with the blocking rule from Bachmair et al (1995). We hope that this paper will inspire other work on theorem proving procedures which inherit information used to limit the search space.

Acknowledgements

I would like to thank the anonymous referees for suggestions which improved the presentation of this paper.

References

1. L. Bachmair and H. Ganzinger. Perfect Model Semantics for Logic Programs with Equality. In *Proc. 8th Int. Conf. on Logic Programming*, pp. 645–659, Cambridge, MA, 1991. MIT Press.
2. L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation Vol. 4, No. 3* (1994) pp. 217–247.
3. L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic Paramodulation. *Information and Computation Vol. 121, No. 2* (1995) pp. 172–192.
4. R. Boyer and J. Moore. A Computational Logic Academic Press, New York (1979).
5. W. Dowling and J. Gallier. Linear–Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *Journal of Logic Programming 3* (1984) pp. 267–284.
6. J. Hsiang and M. Rusinowitch. Proving refutational completeness of theorem proving strategies: The transfinite semantic tree method. *Journal of the Association for Computing Machinery 38* (1991) pp. 559–587.
7. C. Kirchner, H. Kirchner and M. Rusinowitch. Deduction with Symbolic Constraints. *Revue Francaise d'Intelligence Artificielle Vol 4, no. 3* (1990) pp. 9-52.
8. D. Loveland. Mechanical theorem proving by model elimination. *Journal of the Association for Computing Machinery 15(2)* (1968) pp. 236–251.
9. D. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, 1978.
10. C. Lynch, and W. Snyder. Redundancy Criteria for Constrained Completion. *Theoretical Computer Science Vol. 142, No. 2* (1995) 141–177.

11. R. Nieuwenhuis and A. Rubio. Basic Superposition is Complete. In *Proc. European Symposium on Programming*, Lect. Notes in Computer Science, vol. 582, pp. 371–390, Berlin, 1992. Springer-Verlag.
12. R. Nieuwenhuis and A. Rubio. Theorem Proving with Ordering and Equality Constrained Clauses. *Journal of Symbolic Computation Vol. 19 No. 4* (1995) pp. 321–352.
13. R. Nieuwenhuis and A. Rubio. AC-superposition with constraints: No AC-unifiers needed. In *Proc. 12th Int. Conf. on Automated Deduction*, Lect. Notes in Artificial Intelligence, vol. 814, pp. 545-559, Berlin, 1994.
14. J. Pais and G. Peterson. Using forcing to prove completeness of resolution and paramodulation. *J. Symbolic Computation 11* (1991) pp.3-19.
15. G. Peterson. A technique for establishing completeness results in theorem proving with equality. *SIAM Journal of Computing 12* (1983) pp. 82–100.
16. G.A. Robinson and L. T. Wos. Paramodulation and theorem proving in first order theories with equality. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4* pp. 133–150. American Elsevier, New York, 1969.
17. M. Rusinowitch and L. Vigneron. Automated deduction with associative-commutative operators. Internal report 1896, INRIA, May 1993.
18. L. Vigneron. Associative-Commutative Deduction with Constraints. In *Proc. 12th Int. Conf. on Automated Deduction*, Lect. Notes in Artificial Intelligence, vol. 814, pp. 530-544, Berlin, 1994.
19. L. T. Wos, G. A. Robinson, D. F. Carson, and L. Shalla. The concept of demodulation in theorem proving. *Journal of the ACM*, Vol. 14, pp. 698–709, 1967.
20. H. Zhang. *Reduction, Superposition, and Induction: Automated Reasoning in an Equational Logic*. Ph.D. Thesis, Rensselaer Polytechnic Institute (1988).